

i-Eclat: performance enhancement of Eclat via incremental approach in frequent itemset mining

Wan Aezwani Wan Abu Bakar¹, Mustafa Man², Mahadi Man³, Zailani Abdullah⁴

¹Faculty of Informatics and Computing, Universiti Sultan Zainal Abidin, Malaysia

^{2,3}School of Informatics and Applied Mathematics, Universiti Malaysia Terengganu, Malaysia

⁴Faculty of Entrepreneurship and Business (FEB), Centre of Computing and Informatics (CCI),
Universiti Malaysia Kelantan, City Campus, Malaysia

Article Info

Article history:

Received Jul 4, 2019

Revised Jul 28, 2020

Accepted Oct 22, 2019

Keywords:

Association rule mining (ARM)

Dense dataset

Eclat

Incremental-eclat (i-Eclat)

Sparse dataset

Vertical format

ABSTRACT

One example of the state-of-the-art vertical rule mining technique is called equivalence class transformation (Eclat) algorithm. Neither horizontal nor vertical data format, both are still suffering from the huge memory consumption. In response to the promising results of mining in a higher volume of data from a vertical format, and taking consideration of dynamic transaction of data in a database, the research proposes a performance enhancement of Eclat algorithm that relies on incremental approach called an Incremental-Eclat (i-Eclat) algorithm. Motivated from the fast intersection in Eclat, this algorithm of performance enhancement adopts via my structured query language (MySQL) database management system (DBMS) as its platform. It serves as the association rule mining database engine in testing benchmark frequent itemset mining (FIMI) datasets from online repository. The MySQL DBMS is chosen in order to reduce the preprocessing stages of datasets. The experimental results indicate that the proposed algorithm outperforms the traditional Eclat with 17% both in chess and T10I4D100K, 69% in mushroom, 5% and 8% in pumsb_star and retail datasets. Thus, among five (5) dense and sparse datasets, the average performance of i-Eclat is concluded to be 23% better than Eclat.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Wan Aezwani Wan Abu Bakar,

Faculty of Informatics and Computing,

Universiti Sultan Zainal Abidin,

Besut Campus, 22200 Besut Terengganu, 609-6993054, Malaysia.

Email: wanaezwani@unisza.edu.my

1. INTRODUCTION

Data mining (DM) is the research area where the huge dataset in database and data repository are being scoured and mined to find novel and useful pattern. Association analysis is one of the four (4) core data mining tasks besides cluster analysis, predictive modelling and anomaly detection. The task of association rule mining is to discover if there exist the frequent itemset or pattern in database and if any, an interesting relationship between these frequent itemsets can reveal a new pattern analysis for the future decision making. It is a fundamental part of many data mining applications including market basket analysis, web link analysis, genome analysis and molecular fragment mining. In mining frequent itemsets, two (2) main searching strategies could be applied i.e. Horizontal format (breadth first search) or vertical format (depth first search). The searching strategy in mining might give significant effect to overall mining process. Finding frequent itemsets or patterns is a big challenge and has a strong and long-standing tradition in data mining [1] since data

is increasing rapidly in volume [2]. The difficulties arise when new data is added, the naïve approach is to rerun the whole mining algorithm [3] on the whole datasets. Thus, this process results in huge main and cache memory consumption [4]. To the best of our knowledge, there are three (3) basic frequent itemset mining and best-known algorithms, Apriori [1, 2] that lies in horizontal format, FP-Growth [3] and Eclat [4] that lies in vertical format. In this paper, we focus on vertical format by looking deeper on equivalence class transformation (Eclat) algorithm [4] and Eclat-variants algorithm in [5-7] as the research base models. We propose an incremental approach that based on vertical Eclat algorithm and named it as incremental Eclat or i-Eclat to improve the performance of memory. The response time is measured in time execution per seconds.

2. BASIC PRINCIPLES

2.1. Association rule

In the rest of this section, let B be the item base where $B = \{i_1, i_2, \dots, i_m\}$, for $m > 0$ refers to the set of literals called a set of m items. Items may be a product, service, action or atom. A set $X = \{i_1, \dots, i_k\} \subseteq B$ is called an itemset or a k -itemset if it contains k items. A transaction over B is a couple of $T_i = (tid, I)$ where tid is called a transaction identifier and I is an itemset. A transaction $T_i = (tid, I)$ is said to support an itemset $X \subseteq B$ if $X \subseteq I$. A transaction database T is a set of transaction over B . A tidset of an itemset X in T is defined as a set of transaction identifiers in T that support X where $t(X) = \{tid \mid (tid, I) \in T, X \subseteq I\}$. Support of an itemset X in T is the cardinality of its tidset such that support of X is the number of transactions containing X in T , where $sup(X) = |t(X)|$. Illustration of support-confidence framework is given as below:

- The support of rule $X \Rightarrow Y$ is the fraction of transactions in database, D containing both X and Y .

$$Support(X \Rightarrow Y) = \frac{X \cup Y}{|D|}$$

- The confidence of rule $X \Rightarrow Y$ is the fraction of transactions in D containing X that also contain Y .

$$Confidence(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X)}$$

A rule is *frequent* if its support is greater than minimum support (min_supp) threshold. The rules which satisfy minimum confidence (min_conf) threshold is called *strong rule* where min_supp and min_conf are user specified values. An association rule is considered *interesting* if it satisfies both min_supp and min_conf thresholds [8].

2.1. Definitions

Definition 1.

Given a transaction database T over an item base B and a minimal support threshold, s_{min} . The set of all frequent itemsets is denoted by:

$$F(T, s_{min}) = \{X \subseteq B \mid sup(X) \geq s_{min}\}$$

Definition 2. (Candidate itemset):

Given a transaction database T with a minimum support threshold, S_{min} and algorithm for frequent itemset mining of $F(T, s_{min})$, an itemset X is called candidate itemset if the algorithm evaluates if X is frequent or not.

Definition 3. (Intersection):

Let A and B be sets. The intersection of set A and B is denoted by $A \cap B$ is the set containing those elements in both A and B such that

$$A \cap B = \{x \mid x \in A \wedge x \in B\}$$

Definition 4. (Difference set):

Let A and B be sets. The difference of A and B is denoted by $A - B$, is the set containing those elements that are in A but not in B . The difference of A and B is also called the complement of B with respect to A such that

$$A - B = \{x \mid x \in A \wedge x \notin B\}$$

3. TRADITIONAL ECLAT

Eclat is the abbreviation of Equivalence Class Transformation. Its main operation is intersection of tidsets, thus the size of tidsets is one of the main factors affecting the running time and memory usage of Eclat [9, 10]. The bigger tidsets are, the more time and memory are needed [11]. Eclat uses prefix-based equivalence relation, θ_1 along with bottom up search. It enumerates all frequent itemsets. There are two main steps: candidate generation and pruning. In candidate generation, each k -itemset candidate is generated from two frequent $(k-1)$ -itemsets and its support is counted, if its support is lower than the threshold, then it will be discarded, otherwise it is frequent itemsets and used to generate $(k+1)$ -itemsets. Eclat starts with prefix $\{\}$ and the search tree is actually the initial search tree. To divide the initial search tree, it picks the prefix $\{a\}$, generate the corresponding equivalence class and does frequent itemset mining in the sub tree of all itemsets containing $\{a\}$, in this sub tree it divides further into two sub trees by picking the prefix $\{ab\}$: the first sub tree consists of all itemset containing $\{ab\}$, the other consists of all itemsets containing $\{a\}$ but not $\{b\}$, and this process is recursive until all itemsets in the initial search tree are visited [12].

3.1. Candidate generation and pruning

In candidate generation, each k -itemset candidate is generated from two frequent $(k-1)$ -itemsets and its support is counted [13, 14]. If its support is lower than the threshold, then it will be pruned or discarded. Otherwise, it is frequent itemsets and used to generate $(k+1)$ -itemsets. Since Eclat uses vertical layout, counting support is trivial. Depth-first searching strategy is done where it starts with frequent items in the item base and then 2-itemsets from 1-itemsets, 3-itemsets from 2-itemsets and so on. For example, $\{abc\} = \{ab\} \cup \{ac\}$, $\{ab\}$ and $\{ac\}$ are parent of $\{abc\}$. To avoid generating duplicate itemsets, $(k-1)$ -itemsets are sorted in some orders. To generate all possible k -itemsets from a set of $(k-1)$ -itemsets sharing $(k-2)$ items, union operation is conducted of a $(k-1)$ -itemsets with the itemsets that stand behind it in the sorted order, and this process takes place for all $(k-1)$ -itemsets except the last one.

3.2. Equivalence class

An equivalence class $E = \{(i_1, t(i_1 \cup P)), \dots, (i_k, t(i_k \cup P)) | P\}$, considering the set $\{i_1, \dots, i_k\}$ as an item base, it will have a tree of itemsets over this item base and if the prefix P is appended to all itemsets in this new tree, it will have a set of all itemsets sharing the prefix P in the search tree over the item base B . In other word, from this equivalence class, a set of all itemsets sharing the prefix P could be generated and this set forms a sub tree of the initial search tree. We refer traditional Eclat algorithm with *Eclat-tidset*. It starts with prefix $\{\}$ and the search tree is actually the initial search tree. To divide the initial search tree, it picks the prefix $\{a\}$, generates the corresponding equivalence class and does frequent itemset mining in the sub tree of all itemsets containing $\{a\}$, in this sub tree it divides further into two sub trees by picking the prefix $\{ab\}$: the first sub tree consists of all itemset containing $\{ab\}$, the other consists of all itemsets containing $\{a\}$ but not $\{b\}$, and this process is recursive until all itemsets in the initial search tree are visited.

3.3. Horizontal vs. vertical database layout

The works by [15-18] demonstrate how vertical database layout has major advantages over horizontal database layout. Firstly, computing supports of itemsets is much simpler and faster since it involves only intersections of tids and the number of tids automatically indicates the support. In contrast, a complex hash-tree data structures [3] and functions are required for horizontal layout. Secondly, an automatic “reduction” of the database before each scan where only those relevant itemsets to the following scan of the mining process are accessed from disk. In vertical layout, each item i_k in the item base B is represented as $i_k: \{i_k, t(i_k)\}$ and the initial transaction database consists of all items in the item base. For both layouts, it is possible to use the bit format to encode tids and also a combination of both layouts can be used. Figure 1 illustrates horizontal and vertical layout of data representation. Following the depth-first-search of prefix $\{a\}$ as shown in Figure 2, an equivalence class with itemsets $\{ab, ac, ad, ae\}$ will be generated, which are all 2-itemsets containing $\{a\}$. In this sub tree, with the prefix $\{ab\}$, an equivalence class with 3-itemset will be $\{abc, abd, abe\}$. It can be seen that each node in the tree is a prefix of an equivalence class with itemsets right below it. Here, Eclat does not fully exploit the downward closure property because of its depth-first search.

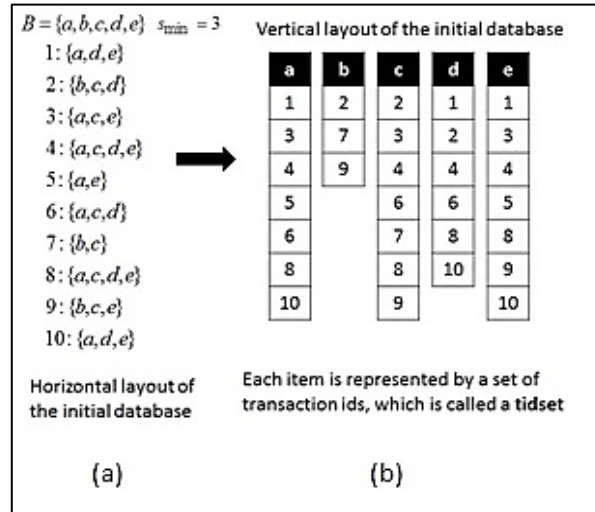


Figure 1. Horizontal and vertical layout

```

Input:  $E((i_1, t_1), \dots, (i_n, t_n)) | P, s_{min}$ 
Output:  $F(E, s_{min})$ 
1: for all  $i_j$  occurring in  $E$  do
2:    $P := P \cup i_j$  // add  $i_j$  to create a new prefix
3:   init( $E'$ ) // initialize a new equivalence class with the new prefix  $P$ 
4:   for all  $i_k$  occurring in  $E$  such that  $k > j$  do
5:      $t_{tmp} = t_j \cap t_k$ 
6:     if  $|t_{tmp}| \geq s_{min}$  then
7:        $E' := E' \cup (i_k, t_{tmp})$ 
8:        $F = F \cup (i_k \cup P)$ 
9:     end if
10:  end for
11:  if  $E' \neq \emptyset$  then
12:    Eclat( $E', s_{min}$ )
13:  end if
14: end for

```

Figure 2. Pseudocode for Eclat algorithm

4. ECLAT VARIANTS

4.1. dEclat algorithm (Eclat-diffset)

The dEclat (different set or diffset) represents an itemset by tids that appear in the tidset of its prefix but do not appear in its tids [5]. In other words, diffset is the difference between two (2) tids (i.e. tidset of the itemsets and its prefix). Using diffset, the cardinality of sets representing itemsets is reduced significantly and this results in faster intersection and less memory usage. An equivalence class with prefix P is considered to contain the itemsets X and Y . Let $t(X)$ denotes the tidset of X and $d(X)$ denotes the diffset of X . When using tidset format, the $t(PX)$ and $t(PY)$ are available in the equivalence class and to obtain $t(PXY)$, the cardinality of $t(PX) \cap t(PY) = t(PXY)$ can be checked [6].

4.2. Sortdiffset algorithm (Eclat-sortdiffset)

The sorting of diffset (sortdiffset) is to enhance dEclat during switching condition [7]. When switching process takes place, there exist tids which do not satisfy the switching condition, thus these tids remain as tids instead of diffset format. The situation results in both tids and diffsets format of itemsets in particular equivalence class and the next intersection process will involve both formats. eclat algorithm.

4.3. Postdiffset algorithm (Eclat-postdiffset)

A study in [19, 20] introduces Postdiffset algorithm that mines in tidsets format for the first level while mining in diffset format in the next level onwards. The performance (in execution time) of mining itemsets are compared between traditional Eclat (tidset) [4], dEclat [5], sortdiffset [7] and postdiffset, and yet, Postdiffset turns to be the third after dEclat and sortdiffset.

5. THE INCREMENTAL APPROACH

5.1. Incremental based on apriori and FP-growth

To improve performance and accuracy of itemset mining, recent researches in [21-25] focus towards parallel and incremental mining approaches. Incremental mining in a dynamic database is established with regards to the itemsets or records of transaction. Incremental in itemsets means an additional new items being added or deleted to the existing itemsets in database whereas incremental in records of transaction means the additional transactions to the existing database transaction. The algorithm for mining weighted maximal frequent itemsets from incremental databases is introduced [26]. By scanning a given incremental database only once, the proposed algorithm extracts a smaller number of important itemsets and provide more meaningful pattern results reflecting characteristics of given incremental databases and threshold settings. A three-way decision update pattern approach (TDUP) is introduced [27]. The approach offers for two support-based measures and synchronization mechanism is periodically triggered to recompute the itemsets offline and results indicate the proposed approach efficient and reliable. The research in [28] proposes incremental parallel apriori-based algorithm on spark (incremental FIM) where it updates frequent itemset based on previous frequent itemset instead of recomputing the whole datasets from scratch. The result shows the algorithm improves mining performance significantly. The IFIN algorithm (a prefix tree structure IPPC-Tree based on FP-Growth is developed [29]. The algorithm maintains a local and changeable order of item in a path nodes from the root to a leaves, which does not waste computational overhead for the previously processed part of data when a new itemset is added or the support threshold is changed and the result depicts an efficient time and memory consumption in mining.

5.2. Incremental based on eclat (i-Eclat)

Our approach is to increment the mining of itemsets from Eclat as our based model. We add with two (2) basic definitions of incremental mining such as follows:

Definition 5. (Incremental Database):

Given a sequence of transaction, T , each of which is called itemset. For a database D and a sequence α , the support of α in D is denoted by $\text{support}_D(\alpha)$ is the frequency of items in D . Suppose that new data, δ is to be added to database D . Then D is said to be original database and δ is the incremental database. The updated database is denoted by $D + \delta$.

Definition 6. (Incremental Records and Itemsets Discovery Problem):

Given an original database D and a new increment to the D which is δ , find all frequent itemsets in database $(D + \delta)$ with minimum possible recomputation and I/O overheads. For each $k \geq 1$, F_k denotes the collection of frequent itemsets of length k in the updated database $(D + \delta)$.

A physical design of i-Eclat is diagrammed in Figure 3. From original database, all frequent items are passed to the first pruning process, getaway **G1**. **G1** is set with the value of min_support threshold prior to generating the list of candidates. To set **G1**, total transaction records are scanned to be multiplied with the percentage of user-specified min_support value. Once the value is obtained, only candidate of frequent itemsets that passed the **G1** value is processed either through Eclat-tidset, Eclat-diffset, Eclat-sortdiffset or Eclat-postdiffset algorithms in Eclat engine. Second pruning process, getaway **2A** or **G2A** where data is written to text file in i-Eclat engine. By writing to text file, candidate itemsets are directed to hard disk storage, so that the resource of memory storage is automatically reduced to enable the processing and executing of full datasets. This is due to maximum memory consumption in handling too many candidate itemsets during intersecting process.

Incremental approach is done in transaction (adding row) or in number of itemsets (adding column). The advantage of an incremental storage structure is that each candidate itemsets in the search space has its counterpart in the database, such that its support can be computed by a few simple database operations, rather than a full scan of a database. The support of a 1-itemset, A is simply the size of column A in the database. So in the first pass of the large set discovery algorithm, it only has to select 1 – itemset whose columns have size above support threshold σ . The support of 2 – itemset, AB is the number of transactions that contain both A and B . Since the tids for A and B are stored in separate column in database, then how many tids would appear in both A and B ? Thus the computation is the intersection between A and B i.e. $A \cap B$. But the problem arises especially in the second pass where many candidates are generated with only very few prove to be large.

For example, there are 600 itemsets out of 1000 itemsets in the first pass are large. Then, in the second pass, the itemsets will be $6002/2 = 180000$ candidates of which only 44! are actually large. Thus, to find these, there are 180000 intersections that consumes over 99% of total database processing time [30].

The terminology *min_supp* used in the pseudocode is to refer to *min_support* threshold value that is determined in terms of percentage where the user-specified *min_supp* value will be divided by 100 and multiply with total rows (records) of each datasets. Next in each loop, starting with the first loop, if the support is greater than or equal (\geq) to *min_supp*, then, getting the *tidset intersection* and *diffset intersection* in *i-Eclat-tidset* and *i-Eclat-diffset* respectively between *i*th column and *i + 1*th column and save to db. In *i-Eclat-sortdiffset*, itemsets are first sorted in descending order depending upon the highest to lowest value of itemset's equivalence class. Then the diffset value between *i*th column and *i + 1*th column will be encountered and save to db while for *i-Eclat-postdiffset*, the first level of looping is based on tidsets process, follows by the second level onwards of looping are getting the result of diffset between *i*th column and *i + 1*th column before saving to db.

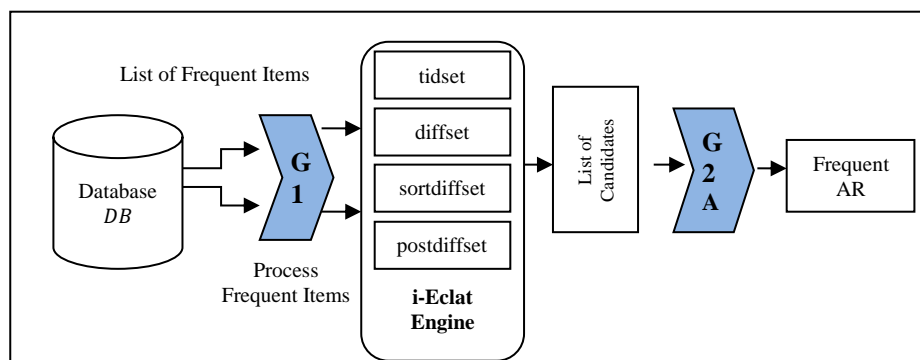


Figure 3. Physical design of i-Eclat engine

6. EXPERIMENTATION

All experiments are performed on a Dell N5050, Intel®Pentium®CPU B960@2.20 GHz with 8 GB RAM in a Win 7 64-bit platform. The software specification for algorithm development is deployed using MySQL version 5.6.20, Apache/2.4.10 (Win32) OpenSSL/1.0.1i PHP/5.5.15 for our web server, php as a programming language. The retrieval of benchmark datasets are from (Goethals, 2003) in a *.dat file format. The two (2) category of datasets are dense (i.e. a dimension with a high probability that one or more data points is occupied in every combination of dimensions) and sparse (i.e. a dimension with a low percentage of available data positions filled). The overall characteristics of benchmark datasets is tabulated in Table 1.

Table 1. Database characteristics

Database	#Size (KB)	# Length (attribute)	#Item	#Records (transaction)	Category
Chess	334	37	75	3196	Dense
Mushroom	557	23	119	8124	Dense
Pumsb_star	1130	48	2088	49046	Dense
Retail	9490	45	16469	88162	Sparse
T10I4D100K	5630	26	1000	100000	Sparse

7. RESULTS AND DISCUSSION

The performance of five (5) dense and sparse datasets is measured based on (1) given. The example of percentage of reduction ratio of *tids* in *B* as compared to *tids* in *A* is calculated based on (1) that determines the outperform percentage of *B*.

$$\frac{(tids\ in\ A) - (tids\ in\ B)}{tids\ in\ A} \times 100\ \% \quad (1)$$

We reveals the experimentation with only taking 50% *min_supp* threshold value that we test for tidset, diffset, sortdiffset and postdiffset algorithms. Figure 4 plots the graph of full chess dataset running in Eclat algorithm and developed i-Eclat algorithm. The i-Eclat outperforms in tidset, diffset and postdiffset with 3%, 32% and 54% respectively lesser in execution time, while Eclat outperforms with 48% in sortdiffset algorithm. Diffset shows a tremendous result in execution time with the least in seconds. The mushroom dataset projects

about a different trend in pattern between Eclat and i-Eclat engine as compared to chess as illustrated in Figure 5. i-Eclat outperforms in tidset, diffset and postdiffset with 74%, 71% and 63% better. But in sortdiffset, it turns to Eclat which performs better with 81 in percentage.

The third dense dataset, pumsb_star shows only a small difference of performance between i-Eclat and Eclat engines. Figure 6 plots the graphs of full pumsb_star such that i-Eclat outperforms only at a small ratio where only 10% is recorded in diffset. This follows by sortdiffset, postdiffset and tidset with only 6%, 3% and 1% better performance. The results of first in sparse category which is retail dataset clearly depicts in Figure 7. The percentage recorded in all algorithms of retail dataset shows a different trend as compared to other dense datasets where Eclat engine outperforms in most of retail dataset in small segments except in full retail dataset. Overall performance evaluation of retail shows that Eclat wins with only 5%, 16%, 9% and 7% better performance in tidset, diffset, sortdiffset and postdiffset respectively. The T10I4D100K dataset is the last in sparse category in this experimentation. Unlike in retail, i-Eclat outperforms in most of the algorithms either in full dataset or in small increment of records and itemsets. Figure 8 depicts the graph where i-Eclat engine seems to outperform with 14% better in tidset, 17% in diffset, only 6% in sortdiffset while the last postdiffset is recorded as 31% better in time execution.

To summarize, the performance of either Eclat engine that is developed by flushing the cache memory (using memory capacity) or i-Eclat engine that is developed by writing to text file (using disk storage capacity), both engines confirms the best algorithm in vertical mining of frequent itemsets belongs to diffset algorithm [5], follows by sortdiffset [7], next is the new developed postdiffset algorithm [19] by the researcher and the last ranking goes to tidset [4]. For the performance of sortdiffset, the results seem to contradict with [7]. Due to MySQL implementation, the time taken for sorting the diffset of itemsets has resulted in longer time execution. To observe on postdiffset algorithm, it turns to outperform in i-Eclat engine with 63% in mushroom, 31% in T10I4D100K and only 3% in both pumsb_star and retail. However, in chess, postdiffset is outstanding in Eclat engine for 54%. For overall datasets, the results indicate that i-Eclat outperform Eclat for 17% both in chess and T10I4D100K, 69% in mushroom, 5% and 8% in pumsb_star and retail datasets. From these percentage figures, dividing with five (5) datasets, average performance of i-Eclat is concluded to be 23.37% better than Eclat engine.

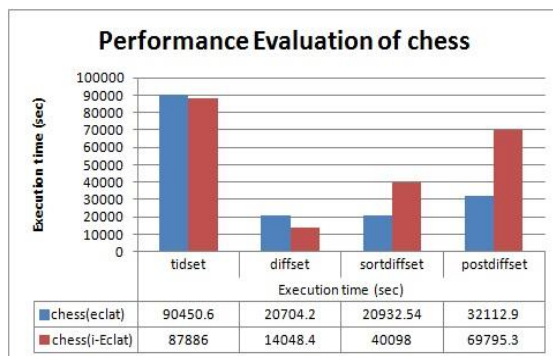


Figure 4. Chess in Eclat engine vs i-Eclat engine

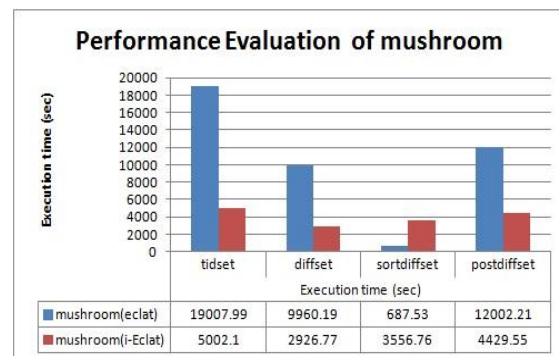


Figure 5. Mushroom in Eclat engine vs i-Eclat engine

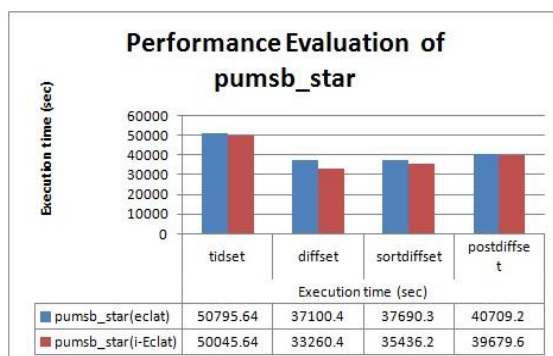


Figure 6. Pumsb_star in Eclat engine vs i-Eclat engine

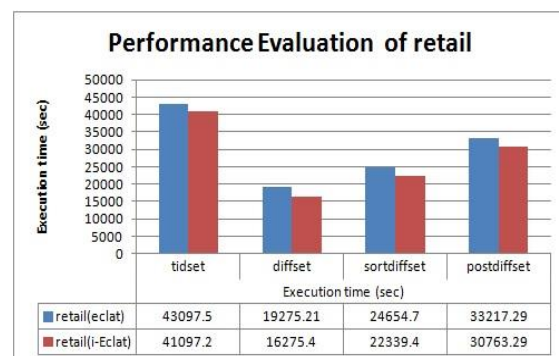


Figure 7. Retail in Eclat engine vs i-Eclat engine

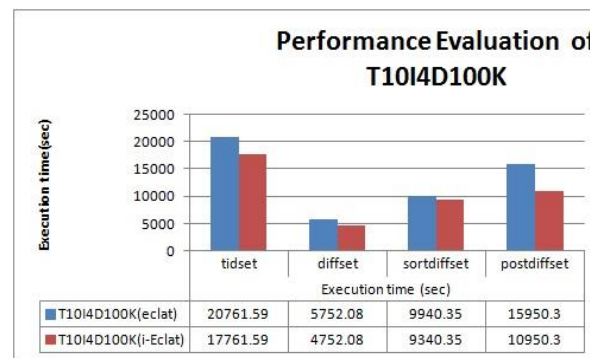


Figure 8. T10I4D100K in Eclat engine vs i-Eclat engine

8. CONCLUSION AND FUTURE DIRECTIONS

The research proves that the more increment in itemset (column) resulting in the more usage of memory as compared to the increment of records of transaction as indicated in Figure 4 to Figure 8. This is due to the increment of itemsets produces the higher cardinality of intersection between each item that needs to be conducted in vertical mining. That is why the much higher execution time can be seen in chess, pumsb_star and retail datasets in spite of mushroom and T10I4D100K datasets. Either Eclat or i-Eclat engine, the performance of both engines is actually depend upon the nature of dataset itself when testing in tidset, diffset, sortdiffset and postdiffset algorithms. However, both engines conforms that among these four (4) algorithms, diffset outperforms other algorithms by certain order of magnitude in all selected datasets. For our next direction in experimentation, we may tackle the issue of infrequent pattern, whether it can fit and provide hidden valuable pattern for analysis.

ACKNOWLEDGEMENTS

Special gratitudes to RMIC of UniSZA for providing financial support under UniSZA internal grant code (UNISZA/2018/DPU/11) and all team members especially teams from UMT and UMK for morale and technical support.

REFERENCES

- [1] Agrawal R, Srikant R., "Fast algorithms for mining association rules," *Proc. 20th int. conf. very large data bases VLDB*, vol. 1215, pp. 487-499, 1994.
- [2] Agrawal R, Imieliński T, Swami A., "Mining association rules between sets of items in large databases". *Proceedings Acm sigmod record*, vol. 22, no. 2, pp. 207-216, 1993.
- [3] Han J, Pei J, Yin Y. "Mining frequent patterns without candidate generation," *Proceedings ACM sigmod record*, vol. 29, No. 2, pp. 1-12, 2000.
- [4] Ogihara Z. P., Zaki M. J., Parthasarathy S., Ogihara M., Li W., "New algorithms for fast discovery of association rules," *3rd Intl. Conf. on Knowledge Discovery and Data Mining*, 1997.
- [5] Zaki M. J., Gouda K., "Fast vertical mining using diffsets," *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 326-335, August 2003.
- [6] Shenoy P., Haritsa J. R., Sudarshan S., Bhalotia G., Bawa M., Shah D., "Turbo-charging vertical mining of large databases," *ACM Sigmod Record*, vol. 29, no. 2, pp. 22-33, May 2000.
- [7] Trieu T. A., Kunieda Y., "An improvement for dEclat algorithm," *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, no. 54, pp. 1-6, February 2012
- [8] Hipp J., Güntzer U., Nakhaeizadeh G., "Algorithms for association rule mining - a general survey and comparison," *SIGKDD explorations*, vol. 2, no. 1, pp. 58-64, July 2000.
- [9] Han J., Cheng H., Xin D., Yan X., "Frequent pattern mining: current status and future directions," *Data mining and knowledge discovery*, vol. 15, no. 1, pp. 55-86, July 2007.
- [10] Borgelt C., "Efficient implementations of apriori and éclat," *FIMI'03: Proceedings of the IEEE ICDM workshop on frequent itemset mining implementations*, December 2003.
- [11] Schmidt-Thieme L., "Algorithmic Features of Eclat," *FIMI*, November 2004.
- [12] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 3, pp. 372-390, May-June 2000.
- [13] Yu X., Wang H., "Improvement of Eclat algorithm based on support in frequent itemset mining," *Journal of Computers*, vol. 9, no. 9, pp. 2116-2123, September 2014.
- [14] Goethals B., "Frequent set mining," *Data mining and knowledge discovery handbook*, Springer, Boston, MA, pp. 377-397, 2005.

- [15] Han J., Pei J., Yin Y., Mao R. "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data mining and knowledge discovery*, vol. 8, no. 1, pp. 53-87, January 2004.
- [16] Borgelt C., Kruse R., "Induction of association rules: Apriori implementation," *Compstat*, Physica, Heidelberg, pp. 395-400, 2002.
- [17] Savasere A., Omiecinski E. R., Navathe S. B., "An efficient algorithm for mining association rules in large databases," Georgia Institute of Technology, 1995.
- [18] Toivonen H., "Sampling large databases for association rules," *VLDB*, vol. 96, pp. 134-145, September 1996
- [19] Bakar W. A. W. A., Jalil M. A., Man M., Abdullah Z., Mohd F., "Postdiffset: an Eclat-like algorithm for frequent itemset mining," *International Journal of Engineering & Technology*, vol. 7, no. 2.28, pp. 197-199, 2018.
- [20] W. A. B. Wan Abu Bakar, Z. B. Abdullah, M. Y. B. Md Saman, M. M. B. Abd Jalil, M. B. Man and T. Herawan, "Vertical Association Rule Mining: Case study implementation with relational DBMS," *International Symposium on Technology Management and Emerging Technologies (ISTMET)*, Langkawai Island, pp. 279-284, 2015.
- [21] Slimani T., Lazzez A., "Efficient analysis of pattern and association rule mining approaches," arXiv preprint arXiv:1402.2892, 2014.
- [22] Man M., Rahim M. S. M., Zakaria M. Z., Bakar W. A. W. A., "Spatial information databases integration model," *International Conference on Informatics Engineering and Information Science*, Springer, Berlin, Heidelberg, pp. 77-90, November 2011.
- [23] Q. Yong, "Integrating Frequent Itemsets Mining with Relational Database," *2007 8th International Conference on Electronic Measurement and Instruments*, pp. 2-543, 2007.
- [24] Ramesh G., Maniatty W., Zaki M. J., "Indexing and Data Access Methods for Database Mining," *DMKD*, June 2002.
- [25] Küng J., Jäger M., Dang T. K., "IFIN+: a parallel incremental frequent itemsets mining in shared-memory environment," *International Conference on Future Data and Security Engineering*, Cham, Springer pp. 121-138, November 2017.
- [26] Yun U., Lee G., "Incremental mining of weighted maximal frequent itemsets from dynamic databases," *Expert Systems with Applications*, vol. 54, pp. 304-327, 2016.
- [27] Li Y., Zhang Z. H., Chen W. B., Min F. "TDUP: an approach to incremental mining of frequent itemsets with three-way-decision pattern updating," *International Journal of Machine Learning and Cybernetics*, vol. 8, no. 2, pp. 441-453, 2017.
- [28] Wen H., Kou M., He H., Li X., Tou H., Yang Y., "A Spark-based Incremental Algorithm for Frequent Itemset Mining," *Proceedings of the 2018 2nd International Conference on Big Data and Internet of Things*, pp. 53-58, October 2018.
- [29] Küng J., Dang T. K., "Incremental frequent itemsets mining with IPPC tree," *International Conference on Database and Expert Systems Applications*, Cham, Springer, pp. 463-477, August 2017.
- [30] Holsheimer M., Kersten M. L., Mannila H., Toivonen H., "A perspective on databases and data mining," *KDD*, vol. 95, pp. 150-155, August 1995.